



NRL/FR/5740--07-10,155

Evolutionary Algorithm Based Automated Reverse Engineering and Defect Discovery

JAMES F. SMITH III

*Surface Electronic Warfare Systems Branch
Tactical Electronic Warfare Division*

September 21, 2007

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 21-09-2007		2. REPORT TYPE NRL Formal Report		3. DATES COVERED (From - To)	
4. TITLE AND SUBTITLE Evolutionary Algorithm Based Automated Reverse Engineering and Defect Discovery				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
6. AUTHOR(S) James F. Smith III				5d. PROJECT NUMBER	
				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory 4555 Overlook Avenue, SW Washington, DC 20375-5320				8. PERFORMING ORGANIZATION REPORT NUMBER NRL/FR/5740--07-10,155	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research One Liberty Center 875 North Randolph Street Arlington, VA 22203				10. SPONSOR / MONITOR'S ACRONYM(S)	
				11. SPONSOR / MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT A data mining based procedure for automated reverse engineering and defect discovery has been developed. The data mining algorithm for reverse engineering uses a genetic program (GP) as a data mining function. A GP is an evolutionary algorithm that automatically evolves populations of computer programs or mathematical expressions, eventually selecting one that is optimal in the sense that it maximizes a fitness function. The system to be reverse engineered is typically a subcomponent of a sensor that may not be disassembled and for which there are no design documents. The sensor is used to create a database of input signals and output measurements. Rules about the likely design properties of the sensor are collected from experts. The rules are used to create a fitness function for the GP, allowing GP-based data mining. This procedure incorporates not only the experts' rules into the fitness function, but also the information in the database. The information extracted through this process is the internal design specifications of the sensor. These design properties can be used to create a fitness function for a genetic algorithm (GA), which is in turn used to search for defects in the digital logic (DL) design. In this report, design flaws in two different sensor systems are detected using a GA. One of these systems makes passive detections, the other makes up part of a radar. In the second case, detecting the flaw allows the design of a radar jamming signal. Uncertainty related to the input-output database and the expert-based rule set can significantly alter the reverse engineering results. This report provides significant experimental and theoretical results related to GP-based data mining for reverse engineering. It presents methods of quantifying uncertainty and its effects. It provides closed-form results that maximize the GP's fitness functions to facilitate understanding of uncertainty. Finally, it examines methods for reducing the uncertainty.					
15. SUBJECT TERMS Data mining Genetic programs Reverse engineering Knowledge discovery Genetic algorithms Defect discovery					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UL	18. NUMBER OF PAGES 30	19a. NAME OF RESPONSIBLE PERSON Dr. James F. Smith III
a. REPORT Unclassified	b. ABSTRACT Unclassified	c. THIS PAGE Unclassified			19b. TELEPHONE NUMBER (include area code) 202-767-5358

CONTENTS

1. INTRODUCTION	1
2. GENETIC ALGORITHMS AND GENETIC PROGRAMS	3
2.1 Genetic Algorithm	3
2.2 Genetic Program	4
3. EVOLUTIONARY ALGORITHM BASED DATA MINING	5
4. THE DIGITAL LOGIC TO BE REVERSE ENGINEERED	5
5. TERMINAL SET, FUNCTION SET, FITNESS FUNCTION, PARAMETERS, AND DATA MINING RESULTS	6
5.1 Terminal and Function Sets	6
5.2 Creating the Database	7
5.3 Calculating the Fitness Based on Rules and Parsimony Pressure	7
5.4 Setting the GP's Parameters.....	9
5.5 Data Mining Results	9
6. AUTOMATED DEFECT DISCOVERY USING A GENETIC ALGORITHM	9
6.1 Automatically Finding Defects in Digital Logic for a Passive Sensor System.....	10
6.2 Genetic-Algorithm-Determined Signals that Characterize a Defect in the Passive Sensor System.....	11
7. AUTOMATIC DEFECT DETECTION APPLIED TO RADAR SIDELOBE BLANKERS	11
8. CONCLUSIONS AND SUMMARY	14
9. ACKNOWLEDGMENTS	15
REFERENCES	16
APPENDIX A Constant False Alarm Rate Algorithm.....	19
APPENDIX B Overall Fitness, Rule Fitness, Input-Output Fitness, Database Structure, and Uncertainty Analysis for GP Data Mining.....	21

EVOLUTIONARY ALGORITHM BASED AUTOMATED REVERSE ENGINEERING AND DEFECT DISCOVERY

1. INTRODUCTION

An engineer has a borrowed system and must determine what design flaws may be present. The owner of the system will not allow it to be disassembled, and the design specifications for the system are not available. The engineer has access to good information about the period in which the system was designed and constructed and about its cost, and from this can infer the parts most likely used. The engineer also knows that the engineers who designed the system were subject to significant cost constraints, so the design is very efficient. Finally, the engineer has access to a historical database of input and output data associated with the system.

Frequently, engineers pursue a trial and error approach to this problem of determining design defects, but this can prove very time consuming if the system is complex. Also, experimental time can be extremely costly; if the engineer can predetermine the most likely input prior to actual experimentation in the lab, then the savings can be enormous. So it would be useful if an automated procedure were available for discovering design flaws.

To solve this problem, a data mining [1] based procedure for automated reverse engineering and defect discovery has been developed and applied to digital logic (DL) examples. The data mining algorithm for reverse engineering uses a genetic program (GP) as a data mining function. A GP is an algorithm based on the biological theory of evolution that automatically evolves populations of computer programs or mathematical expressions, eventually selecting one that is optimal in the sense that it maximizes a measure of effectiveness, referred to as a fitness function [2-5]. The system to be reverse engineered is typically a subcomponent of a sensor, like the system whose design is depicted in Fig. 1. The sensor is used to create a database of input signals and output measurements. Rules about the likely design properties of the sensor are collected from experts. The rules are used to create a fitness function for the GP. GP-based data mining is then conducted [3-5]. This procedure incorporates not only the expert's rules, but also the information in the database, into the fitness function. The information extracted through this process is the internal design specifications of the sensor. The design properties extracted through this process can be used to create a fitness function for a genetic algorithm (GA) [6-15], which in turn is used to search for defects in the sensor design. Unlike a GP, a GA manipulates strings of numbers, not computer programs, with the intent of producing optimal results by maximizing a fitness function. The process of reverse engineering using GP-based data mining followed by using a GA to automatically find defects in the DL discovered by the GP is summarized in Fig. 2. The defect in the DL design is represented by a signal evolved by the GA.

Section 2 of this report introduces GAs and GPs. Section 3 discusses data mining and the use of a GP as a data mining function. Section 4 examines the DL design to be reverse engineered using GP-based data mining. Section 5 explains the GP's terminal set, function set, fitness function, stopping criteria, parameters, and data mining results. Section 6 discusses how a GA can be used to automatically

discover defects in the design of DL for a passive sensor system and includes an example of output. Section 7 discusses GA-based automatic discovery of a flaw in a textbook radar sidelobe blanker design. The output of the GA is a signal that is compared in a summary plot to a mathematically derived signal that represents “truth,” supporting the effectiveness of the GA-based procedure. Section 8 provides conclusions. Appendix A provides mathematical details related to the constant false alarm rate (CFAR) algorithm used in the radar example. Appendix B gives detailed formulations of the rule fitness, fitness score, input-output fitness, and overall fitness for GP-based data mining. It also presents closed-form results for a class of DL maps that provide a global maximum for the rule fitness. From these closed-form results, a short input-output database is derived in closed form to facilitate uncertainty analysis. Appendix B also provides experimental results with detailed descriptions of the evolutionary properties. The class of solutions found in Appendix B is shown to be intrinsically related to the GP’s evolutionary process.

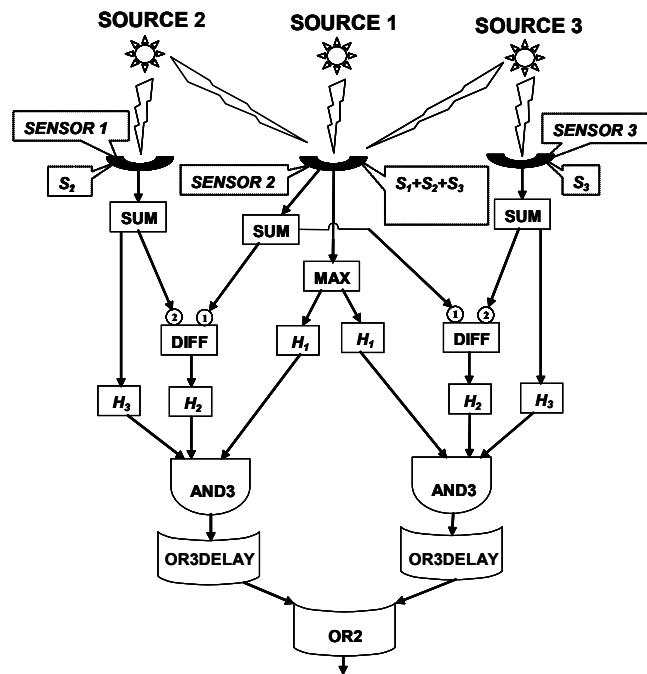


Fig. 1 — A system of three sensors designed to measure signals from source 1 while minimizing the corrupting influences of signals from sources 2 and 3

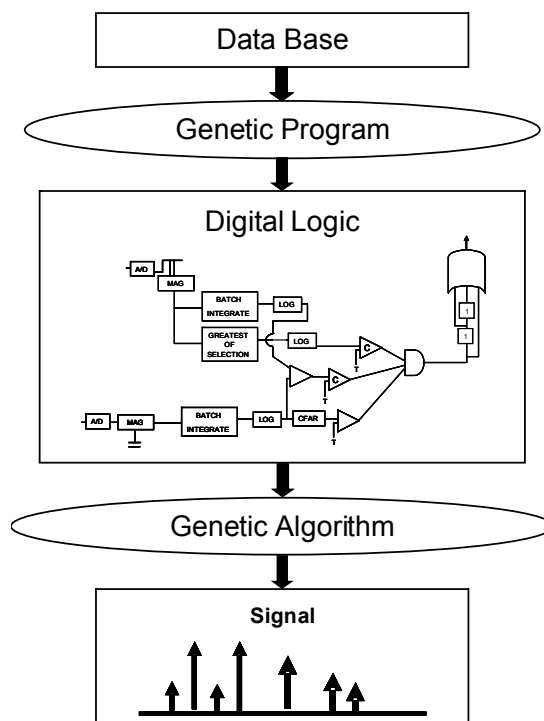


Fig. 2 — Overview of genetic program based reverse engineering and genetic algorithm based automatic defect detection

2. GENETIC ALGORITHMS AND GENETIC PROGRAMS

The following subsections describe GAs and GPs, the differences between the algorithms' structures and applications, and some characteristics specific to the GP used for data mining.

2.1 Genetic Algorithm

A GA is an optimization method that manipulates a string of numbers in a manner similar to how chromosomes are changed in biological evolution [6-15]. An initial population made up of strings of numbers is chosen at random or is specified by the user. Each string of numbers is called a "chromosome" and each numbered slot is called a "gene." A set of chromosomes forms a population. Each chromosome represents a given number of traits that are the actual parameters that are being varied to optimize the fitness function. The fitness function is a performance index that is to be maximized.

The operation of the GA proceeds in steps. Beginning with the initial population, "selection" is used to choose which chromosomes should survive to form a "mating pool." Chromosomes are chosen based on how fit they are (as computed by the fitness function) relative to the other members of the population. More fit individuals end up with more copies of themselves in the mating pool so that they will more significantly affect the formation of the next generation.

Next, two operations are taken on the mating pool: crossover and mutation. First, "crossover" (which represents mating, the exchange of genetic material) occurs between parents. In crossover, a random spot is picked in the chromosome, and the genes after this spot are switched with the corresponding genes of the other parent. Following this, "mutation" occurs. Mutation is a change of the value of a randomly selected gene.

After the crossover and mutation operations occur, the resulting strings form the next generation and the process is repeated. Another process known as “elitism” is also employed. Elitism consists of copying a certain number of fittest individuals into the next generation to make sure they are not lost from the population. Finally, a termination criterion is used to specify when the algorithm should stop, e.g., when a preset maximum number of generations has been reached, when the fitness has gained its maximum possible value, or when the fitness has not changed significantly in a certain number of generations.

2.2 Genetic Program

A GP [2-5] is a problem-independent method for automatically creating graphs that represent computer programs, mathematical expressions, digital circuits, etc. Like a GA, it evolves a solution using Darwin’s principle of survival of the fittest. Unlike the GA, of which it can be considered an extension, its initial, intermediate, and final populations are computer programs.

Like a GA, the individuals that make up the population are subject to selection, crossover, mutation, and elitism. The crossover and mutation operations are constrained to produce structurally valid offspring, i.e., functional computer programs, digital circuits, etc. Two additional GP operations not used with GAs are architecture-altering steps (AAS) [2] and symmetrical replication (SR).

AAS is a method introduced to improve the GP’s convergence. This function randomly changes genes in elite chromosomes, i.e., typically the 50 chromosomes with the highest fitness. For the DL diagram of Fig. 1, AAS might consist of changing H_1 to H_3 or SUM_SIG123 to SUM_SIG2 . This notation is explained in greater detail in sections 4 and 5.

SR refers to replacing a branch entirely with another branch that already existed on the chromosome. SR is similar to mutation; it is not used often and its purpose is to break out of the local fitness landscape, if possible. Like mutation, this replication happens after a specific number of generations have achieved the same maximum fitness. SR is designed to help the GP escape the neighborhood of what may be a local maximum so that the GP can locate the global maximum. Finally, it is generally applied to only a small random set of the elite chromosomes; this allows the GP to better search the space while maintaining diversity in the population.

GPs require a terminal set and function set as inputs. The terminals are the actual variables of the problem. These can include a variable like “x” used as a symbol in building a polynomial, and real constants. The function set consists of a list of functions that can operate on the variables. For example, in a previous application of a GP as a data mining function to evolve fuzzy decision trees symbolically [3-5], the terminal set consisted of fuzzy membership functions and the function set consisted of logical connectives like *AND* and *OR* and logical modifiers like *NOT*. Since fuzzy logic allows more than one mathematical representation of *AND* and *OR*, the function set could be complicated. When the GP is used as a data mining function, a database of input and output information is required. In the case of fuzzy decision trees, the database represented a collection of scenarios about which the fuzzy decision tree to be evolved would make decisions. The database also had entries created by experts representing decisions about the scenarios. The optimal fuzzy decision tree would be the one that could most closely reproduce the experts’ decisions about the scenarios. When the GP is used as a data mining function for evolving DL, the database contains inputs to the DL as well as measured outputs. The experts’ opinions are manifested in the selection of the input and associated output to be included in the database. For the DL case, an additional form of input consisting of “rules” about DL construction is included. The terminal set, function set, database to be data mined, and rules for DL construction are described in greater detail in sections 4 and 5.

3. EVOLUTIONARY ALGORITHM BASED DATA MINING

Data mining is the efficient extraction of valuable non-obvious information embedded in a large quantity of data [1]. Data mining consists of three steps: constructing a database that represents truth; calling the data mining function—e.g., a clustering algorithm, neural net, GA, GP, etc.—to extract the valuable information; and determining the value of the information extracted in the second step, which generally involves visualization.

When used for reverse engineering, the GP typically data mines a database to determine a graph-theoretic structure, e.g., a system's DL diagram or an algorithm's flow chart or decision tree [3-5]. The GP mines the information from a database consisting of input and output values, e.g., a set of inputs to a sensor and its measured outputs. GP-based data mining will be applied to the construction of the DL described in section 4.

To use the GP it is necessary to construct terminal and function sets relevant to the problem. Before the specific terminal and function sets for the reverse engineering problem are described, a more detailed description of the DL to be considered is given in section 4.

4. THE DIGITAL LOGIC TO BE REVERSE ENGINEERED

The first example of a DL to be reverse engineered is depicted in Fig. 1. This DL is not known to the GP. The GP only has access to a database of input signals to the DL and measured output, as well as a collection of rules provided by experts for building the DL.

In Fig. 1, the DL consists of three input channels, each with a sensor attached. The sensors receive signals from sources 1, 2, and 3. Only measurements from source 1, the central source, are of interest. Source 1 is a sufficiently weak emitter that any energy from it that goes into sensors 1 or 3 can be neglected. Unfortunately, sensor 2's measurement may be corrupted by emissions from sources 2 and 3. Fortunately, sources 2 and 3 do not always emit in the same time window as source 1, allowing the output of source 1 to be measured. The DL is constructed so that if there is significant corruption of sensor 2's measurements, the final *OR*-gate returns unity, so the measurements can be ignored.

Figure 1 depicts a number of DL elements that are used repeatedly. DL components and signals will ultimately become elements of the GP's terminal and function sets. The sensors in Fig. 1 will receive an analog signal and convert it to a digital form, i.e., they will map real-valued input to the set of integers. A sampling window of size N is used, i.e., the signal is sampled every Δt seconds for a total of N samples in that window. The sample is indicated by the vector \vec{s}_j in Eq. (1) with sampling beginning at time t_o . The j -subscript implies the signal originates in the j^{th} source, where $j=1,2,3$.

$$\vec{s}_j = [s_j(t_o), s_j(t_o + \Delta t), \dots, s_j(t_o + (N-1) \cdot \Delta t)] \quad (1)$$

The DL function *SUM* given in Eq. (2), represents the logarithmic sum of the absolute value of the time components of the digitized input that has been received for a single window of length N .

$$SUM(\vec{s}_j) = \ln \left[\sum_{k=1}^N |s_j(t_o + (k-1) \cdot \Delta t)| \right] \quad (2)$$

The elements labeled H_i , for $i=1,2,3$, are Heaviside step functions as given in Eq. (3). If the input is greater than or equal to a threshold τ_i , for $i=1,2,3$, then a value of unity is transmitted; otherwise a zero is transmitted.

$$H_i(s) = \begin{cases} 1, & \text{if } s \geq \tau_i \\ 0, & \text{if } s < \tau_i \end{cases}. \quad (3)$$

The DL function *MAX* given in Eq. (4) returns the natural logarithm of the maximum absolute value of the time components of the input signal for a single window of length N . The element labeled *DIFF* takes the difference between input to its first and second terminals as indicated in Eq. (5).

$$MAX(\vec{s}_j) = \ln \left[\bigvee_{k=1}^N |s_j(t_o + (k-1) \cdot \Delta t)| \right] \quad (4)$$

$$DIFF(I_1, I_2) = I_2 - I_1 \quad (5)$$

The DL function *OR3DELAY* takes only Boolean inputs, i.e., it expects zero or one as an input. It waits until it has three consecutive inputs from three consecutive time windows, hence the “3” in its name. Once it receives three consecutive inputs, it yields as an output the maximum of its inputs. Not depicted, but also used in the GP’s function set, is *AND3DELAY*, which takes three inputs of zero or one corresponding to three consecutive time windows and yields as output the minimum of its inputs. *AND2DELAY* uses only two inputs and returns their minimum. Finally, the symbols labeled *AND3*, *OR3*, *AND2*, and *OR2* are the conventional logical connectives *AND* and *OR*, with the numerical designation indicating the number of inputs expected, e.g., *AND3* expects three Boolean inputs.

The signals are additive; at any given time, sensor 2 may record a superposition of the three sources’ transmissions, which is represented by $s_1(t) + s_2(t) + s_3(t)$. If the three sensors’ signals are of sufficient magnitude, the measurement is considered corrupted and the final *OR* in Fig. 1 returns unity.

5. TERMINAL SET, FUNCTION SET, FITNESS FUNCTION, PARAMETERS, AND DATA MINING RESULTS

This section describes the terminal set, function set, database construction, fitness function, the GP’s essential parameters, and the results of data mining for the DL represented by Fig. 1. The description is given in terms of DL elements and properties, but the GP-based reverse engineering technique is very general and can be applied to any system that can be described in a graph-theoretic language, e.g., decision processes described in terms of decision trees [3-5].

5.1 Terminal and Function Sets

The terminal set consists of the following elements:

$$T = \{SUM_SIG123, MAX_SIG123, SUM_SIG2, MAX_SIG2, SUM_SIG3, MAX_SIG3\} \quad (6)$$

where

$$SUM_SIG123 = SUM(\vec{s}_1 + \vec{s}_2 + \vec{s}_3), \quad (7)$$

$$MAX_SIG123 = MAX(\vec{s}_1 + \vec{s}_2 + \vec{s}_3), \quad (8)$$

$$SUM_SIG2 = SUM(\vec{s}_2), \quad (9)$$

$$MAX_SIG2 = MAX(\bar{s}_2), \quad (10)$$

$$SUM_SIG3 = SUM(\bar{s}_3), \quad (11)$$

and

$$MAX_SIG3 = MAX(\bar{s}_3). \quad (12)$$

All sensor measurements begin at time t_o .

The function set consists of the following elements:

$$F = \{AND3, OR3, AND2, OR2, AND3DELAY, OR3DELAY, H_1, H_2, H_3, DIFF\}. \quad (13)$$

The function *AND3DELAY* is not used in Fig. 1. By including it, the GP's ability to discriminate against extraneous functions is emphasized.

The GP's goal is to evolve a solution that represents Fig. 1. The GP's ability to do this will be determined largely by the fitness function and the underlying databases, discussed below. The chromosome to be evolved by the GP in prefix notation that represents Fig. 1 is given in Eq. (14):

$$\begin{aligned} &OR2\ OR3DELAY\ AND3\ H_3\ SUM_SIG3\ H_2\ DIFF\ SUM_SIG3\ SUM_SIG123\ H_1\ MAX_SIG123 \\ &OR3DELAY\ AND3\ H_3\ SUM_SIG2\ H_2\ DIFF\ SUM_SIG2\ SUM_SIG123\ H_1\ MAX_SIG123. \end{aligned} \quad (14)$$

5.2 Creating the Database

The database used for constructing the fitness function consists of inputs to the system to be reverse engineered and measured output. For the DL diagram of Fig. 1, the input is a collection of signals from sources 1 through 3 and the associated measured output. The output of the DL represented by Fig. 1 will consist entirely of zeros and ones. The fact that the system maps real-valued input into the doublet set $\{0, 1\}$ complicates the reverse engineering process. For a case like this, the rule set used to construct the fitness function is of paramount importance if a unique solution is to be found. These rules correspond to conditions provided by human experts regarding what elements the logic may have, how they are interconnected, and their abundance.

5.3 Calculating the Fitness Based on Rules and Parsimony Pressure

This subsection discusses a few of the rules used to construct the fitness function. Typically, each rule represents a desirable feature that the final solution must have or might have. Toward this goal, candidate solutions, i.e., chromosomes within the GP's evolving population, are given a higher fitness for showing characteristics consistent with the rules.

5.3.1 Rule Examples and Their Rewards

For the DL depicted in Fig. 1, a number of rules can be developed without knowing the specific design. For example, when actually using the DL or upon close analysis of input-output relations it is observed that the logic waits for a period corresponding to three consecutive time windows before making a declaration. Based on the logic's function the following rule is formulated:

Rule 1: There must be a three-time-window time delay built into the DL.

Another rule provided by experts is

Rule 2: The DL must have a delay near the output level.

Since data and expert intuition seem to indicate that there is an *AND* or *OR* at the end, this rule follows:

Rule 3: The last operation is a Boolean.

It follows from the properties of the hardware implementation of the Heaviside step function and the *DIFF* operator that

Rule 4: The Heaviside step function must take an integer input;

and

Rule 5: The *DIFF* operator must have a two-signal input.

Likewise, Rule 6 represents common sense:

Rule 6: It is unlikely that there is a functional composition of Heaviside step functions.

In each case when the GP determines that a candidate solution satisfies the rule, the fitness for that solution is incremented. Other rules were used, but the above set captures the spirit of how rules are obtained and rewards are awarded. So it is observed that some of the rules arise from an observation (like Rule 1), but others arise from properties of the logic elements (Rule 3) or from common sense (Rule 6). The rules as presented here have been simplified for ease of understanding. Appendix B provides a more detailed description of the rules actually used.

5.3.2 Controlling Bloat

One of the fundamental problems associated with genetic programming is controlling bloat. Bloat [16] refers to excessive growth of candidate solutions. It is found empirically that every 50 generations, the length of candidate solutions, i.e., chromosomes within a GP's evolving population, increases by a factor of three. Various procedures have been invented for controlling bloat. Two of the most popular are the Koza depth limit [2] and parsimony pressure [3-5, 16]. Another technique has been invented recently that involves the use of computer algebra to control bloat when using a GP to evolve mathematical expressions [3]. This algebraic approach was not used for the study described here but may be applied in a future extension of this work.

The procedure applied for evolving the DL of Fig. 1 was parsimony pressure. The parsimony pressure can be regarded as a numerical representation of Occam's razor [17]. Just as Occam's razor favors the simpler of two hypotheses that explain a data set, parsimony pressure favors the simpler, i.e., the shorter of two DL designs with the same basic fitness. The basic fitness is the fitness determined by the rules and input-output database. The parsimony pressure for a given chromosome is calculated by multiplying the parsimony coefficient, α , by the length of the chromosome. The overall fitness is determined by subtracting the parsimony pressure from the basic fitness. Thus given two candidate solutions with the same basic fitness, the shorter of the two will have the higher overall fitness.

5.4 Setting the GP's Parameters

The GP requires the specification of many parameters for efficient operation. Modification to these parameters can have a significant impact on the convergence time and solution found by the GP. Some of the most important parameters are population size, database size, probability of mutation, probability of crossover, and parsimony pressure.

The size of the population of evolving chromosomes is very important. A large population can represent a significant degree of diversity, potentially resulting in the determination of a better solution. Increased population size can also have a significant effect on run time. The default population size used for the DL depicted in Fig. 1 is 1000.

Increasing the size of the database can have a significant effect. A larger database may contribute to the determination of a unique solution or a smaller final class of high fitness potential solutions. Unfortunately, given that the DL in Fig. 1 maps from real-valued inputs to the doublet set $\{0, 1\}$, the benefits of a larger database are limited. It becomes important to embed high-quality rules in the fitness function, rewarding good candidate solutions, i.e., those consistent with the rules, and penalizing poor quality chromosomes.

The probabilities of crossover and mutation can also be very important. Low probabilities of crossover and mutation can result in only limited regions of the fitness landscape being explored, i.e., the GP will get locked into a local neighborhood and not escape.

Two other parameter-dependent operations are AAS and SR, which are applied to the elite chromosomes, i.e., the top 50 with the highest fitness. When applying AAS, the GP selects about 63% of the elite chromosomes in each generation. Likewise, SR is applied to 7% of the elite chromosomes that have been through 12 generations where the maximum fitness has not changed.

Appendix B gives detailed formulations of the rule fitness, fitness score, input-output fitness, and overall fitness for GP-based data mining. Closed-form results for a class of DL maps that provide a global maximum for the rule fitness are given. From these closed-form results a short input-output database is derived in closed form to facilitate uncertainty analysis.

5.5 Data Mining Results

The GP successfully discovered the DL of Fig. 1 through data mining after 60 generations using the parameters given in the previous subsection. The SR operation proved to be very effective and helped accelerate convergence significantly.

Appendix B provides experimental results with detailed descriptions of the evolutionary properties. The class of solutions found in Appendix B is shown to be intrinsically related to the GP's evolutionary process.

6. AUTOMATED DEFECT DISCOVERY USING A GENETIC ALGORITHM

The second phase of this work, automatic defect detection, involves use of a GA instead of a GP. The automatic defect detection methodology will use a GA to automatically evolve signals that characterize design defects. Like a GP, the GA requires a fitness function. The DL automatically determined by the GA will be used to construct the GA's fitness function.

6.1 Automatically Finding Defects in Digital Logic for a Passive Sensor System

The function of the DL in Fig. 1 is to determine if emissions from sources 2 and 3 are corrupting the measured emission from source 1. If only data from source 1 is measured, the DL is expected to return a zero, but if sensor 2's measurements are corrupted by contributions from sources 2 and 3, then the DL returns unity. If the output of the DL is unity, the measurement is discarded. The intention of the design engineers was that the DL would allow relatively uncorrupted measurements of the output of source 1 to be recorded.

Given the design indicated by Fig. 1, if the original design engineers were not careful, it might be possible for sensors 1 and 3 to receive signals \bar{s}_2 and \bar{s}_3 of sufficiently low magnitude that the DL does not recognize them as corruption. At the same time, they obscure the characteristics of the desired signal \bar{s}_1 broadcast by source 1 and received by sensor 2. Such an event would represent a significant defect. It is this defect that the GA actually found. The defect found by the GA reflects the fitness function used. It is possible for the GA to find many other defects given other fitness functions.

The defect signal described above that will escape detection by the DL takes the form

$$\bar{S}_D = \bar{s}_1 + \bar{s}_2 + \bar{s}_3, \quad (15)$$

such that there is so little variation in the magnitude of \bar{S}_D that the signature form of \bar{s}_1 is no longer observed. One way of potentially satisfying the constraints is to look for a superposition signal, \bar{S}_D , such that it is flat, i.e., it has no variation in magnitude and the signals \bar{s}_2 and \bar{s}_3 do not result in a declaration of unity by the DL.

The DL reverse engineered by the GP yields unity if the signals \bar{s}_2 and \bar{s}_3 are of sufficient magnitude during any three consecutive time windows. In this section, it is assumed that each time window has a length 5; thus three intervals give a time window of length 15. The selection of a time window of length 5 was made to provide a simple example. It is assumed that the form of signal \bar{s}_1 broadcast by source 1 is known, whereas the signals broadcast by sources 2 and 3 are unknown and may vary in time.

The GA will attempt to find signals \bar{s}_2 and \bar{s}_3 such that \bar{S}_D shows little variation over three consecutive time windows, while at the same time the DL declares zero. The fitness function includes rules to facilitate finding this solution. It is important to observe that the DL is built into the fitness function and used for numerical calculation.

The chromosomes for this application represent the values of candidate solutions for \bar{s}_2 and \bar{s}_3 . It is not necessary to solve for \bar{s}_1 , since it is assumed to be known. Also, unlike GP-based data mining, there is no database required for automatic defect discovery.

The chromosome size is $2 \times \text{length (time window)} \times 3$, or 30. The factor of two arises because the chromosome represents both \bar{s}_2 and \bar{s}_3 ; the factor of three, because measurements from three consecutive time windows are included for both signals. The three time window measurements of \bar{s}_2 are encoded in the first half of the chromosome, positions 1 to 15. The second half of the chromosome, positions 16 to 30, represents \bar{s}_3 . Chromosomes were selected to represent both signals to allow genetic information to be exchanged between the signals through crossover.

The GA uses the following parameter values: the population size is 1000, chromosome size 30, the probability of crossover is 90 percent, and probability of mutations 10 percent. The elitism size, i.e., the number of highest fitness individuals automatically retained from the previous generation, is 50.

The GA uses as a stopping criterion that the fitness must exceed a threshold, 0.99, or that 1000 generations have passed. These parameters were determined based on experience with the DL problem and GAs.

6.2 Genetic-Algorithm-Determined Signals that Characterize a Defect in the Passive Sensor System

When GA-based automatic defect detection is applied to the DL of Fig. 1 using the defect criterion of the previous section, the program converges after about 300 generations. The following assumption is made: over three consecutive time windows the signal received by sensor 2 due to source 1 is $[-1, 2, -3, 20, -4, -2, 18, -1, 1, 2, -1, -2, 17, -1, 1]$. For the assumed value of sensor 2's measurements the GP found the following solution for \bar{s}_2 and \bar{s}_3 over the same three consecutive time windows:

$$\text{sensor 1 measurements} = [-10, -22, 24, 31, 58, 41, -51, -17, -46, -30, -25, -10, 13, -29, 9] \quad (16)$$

and

$$\text{sensor 3 measurements} = [51, -20, 19, -11, -14, 1, -7, -22, 5, -12, -14, -28, 10, -10, 30]. \quad (17)$$

It is observed that for the chromosome found, the DL returns zero and the 15 elements of \bar{S}_D are all 40 in magnitude, i.e., the magnitude of \bar{S}_D shows no variation over the three time windows. Thus the GA has found a significant design flaw in the DL. It is readily observed that this example is very simple. It was included to illustrate the potential of the GA-based procedure for automatically finding flaws in designs.

7. AUTOMATIC DEFECT DETECTION APPLIED TO RADAR SIDELobe BLANKERS

A more sophisticated application of automatic defect detection is to the theory of sidelobe blankers (SLBs) in radar design. In Fig. 3, radar O desires to detect platform A. Jamming platform B protects A by injecting energy into O's sidelobe. R_A is the range of platform A from radar O's main antenna. R_B is the range of platform B from radar O's auxiliary antenna. (The figure does not attempt to preserve scale.)

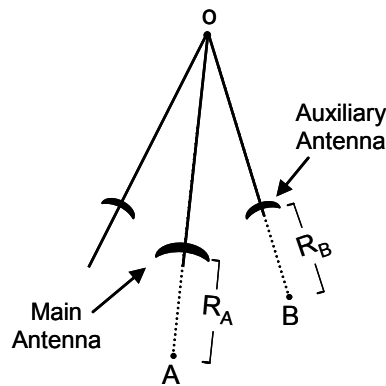


Fig. 3 — Radar O, platform to be protected A, and jammer B.

The main beam of the radar is used to detect platforms of interest, e.g., a ship or an airplane. The radar antenna pattern [18] also has sidelobes. An enemy platform may carry a jamming device designed to put energy into radar O's sidelobe to produce a false target, i.e., a false image on the radar O's output screen. SLBs are designed to defeat this type of countermeasure. The fundamental design requirements for a sidelobe blanker are described in [19]:

“Generation of false targets in different directions, other than that of the jammer-carrying aircraft, requires injecting jamming signals into the radar's sidelobes. Many radars employ SLBs to defeat this type of electronic counter measure. A sidelobe blanker compares the signal detected in its main channel against a signal detected in an auxiliary antenna channel. The response of the auxiliary antenna channel is arranged to be greater than any main antenna sidelobe response in that direction but less than the antenna's main-lobe response. A signal is blanked if its auxiliary antenna response is greater than the main antenna response, which signifies that a sidelobe jammer is present.”

The description above is enough to create the necessary DL design for the SLB. It is useful if an additional commonsense constraint is imposed. This constraint is as follows: if the magnitude of the signal along the auxiliary antenna exceeds a certain threshold, this should be a condition for blanking. If this signal is of large magnitude, then it is obvious that jamming is occurring.

Typically when a signal is blanked by the DL, the input measurement is replaced by thermal noise. It might be desirable that something more sophisticated be carried out, but in the classic textbook design described by [19], this is what is done.

With the design specifications provided by the quote above and a commonsense observation, it is possible to produce the necessary DL. A DL design for a SLB arising from these constraints is given in Fig. 4. The formalism in Fig. 4 has the same meaning as in Fig. 1 with the exception that $a*SUM$ and $b*MAX$ imply that the SUM and MAX functions used in Fig. 1 are multiplied by parameters a and b . Also, there are two additional symbols. The ellipse with the $CFAR$ label represents the CFAR algorithm, explained below. The circle on the edge between H_I and $AND3$ represents the modifier NOT .

Platforms A and B desire to exploit design defects to cause radar O's DL to blank inappropriately, i.e., to blank and subsequently cause the radar O to replace the return from platform A with thermal noise. This will deprive radar O's operator of knowledge of platform A's presence.

As with the previous example, it is necessary to construct the fitness function for the GA so that the jamming signal, i.e., the signal that represents the design defect, can be automatically evolved by the GA. Two different sources of information are used to construct the fitness function. The first is the DL that was reverse engineered by GP-based data mining. The second is human expertise about what sort of jamming is to occur.

Just as in the case of reverse engineering the DL in Fig. 1, when GP-based data mining is applied to Fig. 4, a collection of rules about the likely components used in the SLB are required. Likewise, a database of measurements is required. If the rules related to the SUM function and the MAX function are replaced by $a*SUM$ and $b*MAX$, then a set of rules similar to those used for Fig. 1 can be used for Fig. 4. Finally, the input-output database is created in a straightforward fashion as for the DL design in Fig. 1.

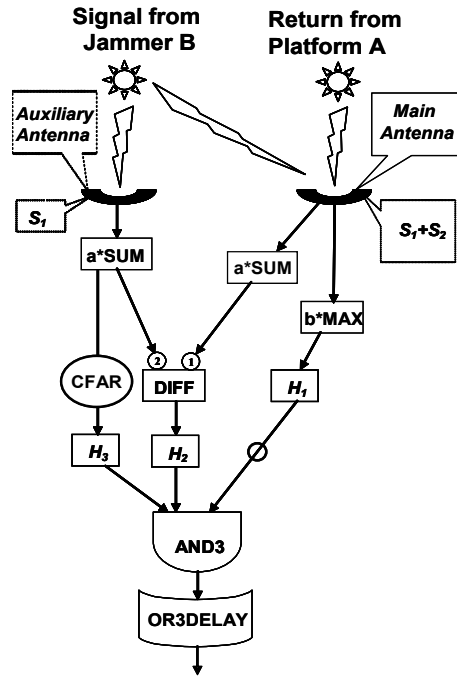


Fig. 4 — The sidelobe blanker digital logic

Also, the GP-based data mining process requires a terminal set and function set. Terminal and function sets similar to those used for Fig. 1 can be used for Fig. 4 if the same substitutions for *SUM* and *MAX* are made along with the additions of *CFAR* and *NOT* modules to the function set.

The *CFAR* algorithm determines if the amplitude in a test cell is high enough and then slides on to another test cell. It does this as follows. The two range cells adjacent to the test cell are referred to as guard cells. When testing, the *CFAR* algorithm takes the m range cells just before the guard cell preceding the test cell, eliminates the maximum amplitude cell from the m cells, then averages over the remaining $m-1$ cells. The *CFAR* algorithm calculates a similar average for the m range cells after the guard cell on the other side of the test cell.

The *CFAR* algorithm takes the maximum of the two averages and compares them to the test amplitude. If the test amplitude exceeds the maximum of the average amplitudes by a certain amount, then the test amplitude is passed on to the next processing step. Otherwise, the test amplitude and the two guard amplitudes are replaced by the average thermal noise. Afterward, the test window slides on to the next test cell. Appendix A provides additional mathematical detail related to the *CFAR* algorithm.

It should be noted that it is possible to have the GP construct the *CFAR* module from more basic components. Although this was not carried out for this example, it may be pursued in the future.

After making the above substitutions into the rules, terminal set, and function set, and creating the input-output database, GP-based data mining was conducted. The DL in Fig. 4 was readily found. This process generally required less than two hours of run time on a 2 GHz laptop processor. The same GP was used as when the DL design in Fig. 1 was reverse engineered.

Following the reverse engineering of the DL design, the next step in constructing the GA's fitness function is determining a rule about what sort of defect is to be discovered. The defect in the DL's design

to be exploited is that it may be possible for platform B to make it blank through jamming when the return from platform A has been received. From looking at the DL design in Fig. 4 it is observed that if blanking occurs for range cells before or after the range cell that contains platform A, then platform A's range cell will be blanked also. It follows that platforms A and B desire to use the GA to design a signal to exploit this defect, if possible.

So the GA's fitness function combines the above goal and the DL design. This fitness function was subsequently created. It was found that within about 65 GA generations that the necessary signal was created. This experiment was conducted many times with the same result. It is interesting to observe that for the DL design given in Fig. 4, it is possible to mathematically derive a closed-form-exact solution for the necessary jamming signal. This signal was derived mathematically and used as truth to show that the GA had converged to the proper signal. A summary of these results is given in Fig. 5.

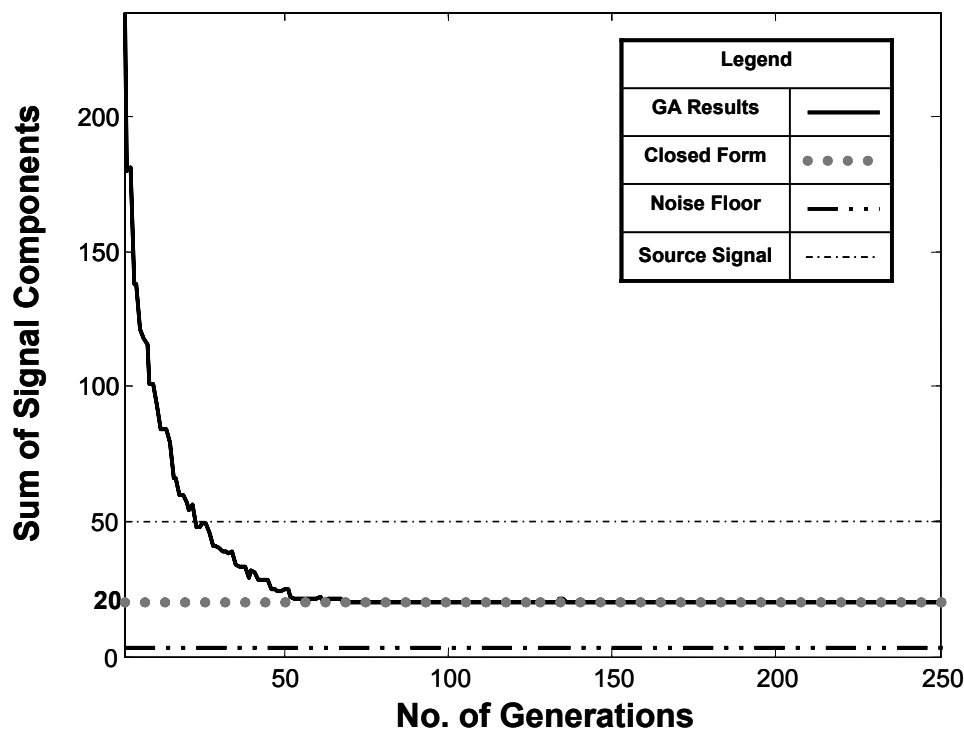


Fig. 5 — Numerical GA result for sidelobe blanker technique versus closed-form result

8. CONCLUSIONS AND SUMMARY

Given a system that may not be disassembled and inspected, and for which the design specifications are unavailable, it can be difficult to determine the design of its underlying digital logic (DL). A genetic program (GP) has been used as a data mining function to reverse engineer DL. The databases that were subjected to data mining consisted of known input to the DL and the associated measured output. A set of rules provided by experts related to the DL was also used to guide the evolutionary process.

The GP has been proven to be capable of reverse engineering complicated DL designs. It is found that having a set of expert rules is essential; the measured output of the DL is rarely sufficient to uniquely reverse engineer the design.

Frequently, part of the motivation for reverse engineering DL is to understand potential errors the design can introduce into measurement systems. To automate the discovery of design defects, a genetic algorithm (GA) based procedure has been developed. This procedure uses the DL design discovered with the GP as part of the fitness function for the GA. In addition to DL making up part of the fitness function, rules about the design's expected output during use are incorporated into the GA's fitness. For the DL examples in this report that the GP reverse engineered, the GA was able to find significant defects within 125 generations for one example and 300 generations for another.

Two GA-based automatic defect detection examples are presented in this report. The first is for a passive sensor system and the second is related to finding a radar design flaw in a textbook logic design. For the passive sensor case, corrupting signals were found that contaminated measurements. For the radar case, the GA found a signal that will cause the radar to blank, i.e., to lose information about returns from a detected platform. From the radar designer's point of view this is a significant design flaw. The radar example permits a closed-form-exact mathematical solution for the signal to be derived. The derived signal represents a measure of absolute truth for this example. The GA found a solution that agreed with the closed-form result within 65 generations.

This report presents explicit formulations of fitness functions and rules. Closed-form results for a class of optimal solutions to the rule fitness are shown to be effective in understanding required input-output database properties. The closed-form results are also extremely useful in understanding the effect of rule or input-output database uncertainty on the final solution evolved by the GP.

Experimental observation and theoretical analysis of the effects of uncertainty show that even when there is a significant reduction in the quality of rule or input-output measurement information, the DL map evolved by the GP will still carry enough information for the design of signals with specific properties. The creation of these signals is considered of greater importance than having the exact DL design for the sensor device. So the results of the uncertainty analysis related to the closed-form expressions are deemed of great importance.

This report provides significant experimental results and theoretical analysis to support the above conclusions. It provides reverse engineered DL examples along with details of their evolutionary history and the implications of uncertainty.

9. ACKNOWLEDGMENTS

This work was sponsored by the Office of Naval Research. The author would also like to acknowledge Dr. Jeffrey Heyer for support and useful advice. Gratitude is extended to Mr. ThanhVu Nguyen for providing useful advice and programming support.

REFERENCES

1. J.P. Bigus, *Data Mining with Neural Networks* (McGraw-Hill, New York, 1996), Ch. 1.
2. J.R. Koza, F.H. Bennett III, D. Andre, and M.A. Keane, *Genetic Programming III: Darwinian Invention and Problem Solving* (Morgan Kaufmann, San Francisco, 1999), Ch. 2.
3. James F. Smith III, "Fuzzy logic resource manager: real-time adaptation and self-organization," *Signal Processing, Sensor Fusion, and Target Recognition XIII*, I. Kadar, ed., Vol. 5429, pp. 77-88, SPIE Proceedings, Orlando, 2004.
4. James F. Smith III, "Fuzzy logic resource manager: decision tree topology, combined admissible regions, and the self-morphing property," *Signal Processing, Sensor Fusion, and Target Recognition XII*, I. Kadar, ed., Vol. 5096, pp. 104-114, SPIE Proceedings, Orlando, 2003.
5. James F. Smith III, "Fuzzy Logic Resource Manager: Evolving Fuzzy Decision Tree Structure that Adapts in Real-Time," *Proceedings of the Sixth International Conference on Information Fusion (FUSION 2003)*, Cairns, Australia, Vol. 2, pp. 838-844, International Society of Information Fusion, 2003.
6. D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning* (Addison-Wesley, Reading, Massachusetts, 1989).
7. J.F. Smith III and R.D. Rhyne II, "A Resource Manager for Distributed Resources: Fuzzy Decision Trees and Genetic Optimization," *Proceedings of the International Conference on Artificial Intelligence, IC-AI'99, Las Vegas*, H. Arabnia, ed., Vol. 2, pp. 669-675, CSREA Press, 1999.
8. James F. Smith III and J. Andrew Blank; "Co-evolutionary data mining for fuzzy rules: automatic fitness function creation, phase space, and experiments," *Data Mining and Knowledge Discovery: Theory, Tools, and Technology*, B.V. Dasarathy, ed., Vol. 5098, pp. 59-70, SPIE Proceedings, Orlando, 2003.
9. James F. Smith III, "Co-evolutionary Data Mining to Deal with Adaptive Adversaries," *Proceedings of the International Conference on Machine Learning; Models, Technologies and Applications (MLMTA'03)*, Las Vegas, H.R. Arabnia and E.B. Kozerenko, eds., pp. 3-9, CSREA Press, 2003.
10. James F. Smith III, Robert D. Rhyne II, and Kristin Fisher, "Data mining for multiagent rules, strategies, and fuzzy decision tree structure," *Data Mining and Knowledge Discovery: Theory, Tools, and Technology IV*, B.V. Dasarathy, ed., pp. 386-397, SPIE Proceedings, Orlando, 2002.
11. James F. Smith III, "Fuzzy Logic Resource Management in a Multi-agent Adversarial Environment," *Proceedings of the 2002 International Conference on Information and Knowledge Engineering (IKE'02)*, Las Vegas, H. Arabnia, Y. Mun, and B. Prasad, eds., pp. 234-240, CSREA Press, 2002.
12. James F. Smith III, "Genetic Program Based Data Mining to Discover Fuzzy Rules," *Proceedings of the International Conference on Artificial Intelligence, IC-AI'02, Las Vegas*, H.R. Arabnia and Y. Mun, eds., pp. 502-508, CSREA Press, 2002.
13. J.F. Smith III and R.D. Rhyne, II, "Genetic Algorithm Based Optimization of a Fuzzy Logic Resource Manager: Data Mining and Co-evolution," *Proceedings of the International Conference on Artificial Intelligence, IC-AI'2000, Las Vegas*, H.R. Arabnia, ed., Vol. 1, pp. 421-428, CSREA Press, 2000.

14. J.F. Smith III and R.D. Rhyne II, "Fuzzy logic resource manager: tree structure and optimization," *Signal Processing, Sensor Fusion, and Target Recognition X*, I. Kadar, ed., Vol. 4380, pp. 312-323, SPIE Proceedings, Orlando, 2001.
15. James F. Smith III, "Fuzzy logic resource manager: multi-agent techniques, fuzzy rules, strategies, and fuzzy decision tree structure," *Signal Processing, Sensor Fusion, and Target Recognition XI*, I. Kadar, ed., Vol. 4729, pp. 78-89, SPIE Proceedings, Orlando, 2002.
16. S. Luke and L. Panait, "Fighting Bloat With Nonparametric Parsimony Pressure," *Parallel Problem Solving from Nature—PPSN VII. 7th International Conference, Granada, Spain, 2002, Proceedings*, J.J.M. Guervos et al., eds. LNCS Vol. 2439, pp. 411-421 (Springer-Verlag, Berlin, 2002).
17. J. Gribbin, *Companion to the Cosmos* (Orion Publishing Group Ltd, London, 1996), p. 299.
18. L.V. Blake, *Radar Range-Performance Analysis* (Artech House, Boston, 1986).
19. D.C. Schleher, *Electronic Warfare in the Information Age* (Artech House, Boston, 1999), Ch. 1, pp. 227-228.

Appendix A

CONSTANT FALSE ALARM RATE ALGORITHM

The constant false alarm rate (CFAR) algorithm selects a test range cell and one guard cell on each side. The CFAR algorithm eliminates the maximum amplitude from the right-most m cells, $r_1^R, r_2^R, \dots, r_m^R$, and the left-most cells, $r_1^L, r_2^L, \dots, r_m^L$. The remaining $m-1$ to the right are averaged, and likewise for the remaining $m-1$ to the left.

The CFAR algorithm takes the maximum of the two averages and compares them to the test amplitude. If the test amplitude is sufficiently higher than the maximum of the average amplitudes, then it is passed on to the next processing step. Otherwise, the test amplitude and the two guard amplitudes are replaced by the average thermal noise. After the test, the window slides on to the next test case.

Let

$$\Delta r = \text{radar range resolution} \quad (\text{A1})$$

and r_k be the range denoted by the k^{th} range cell

$$r_k = (k-1)\Delta r. \quad (\text{A2})$$

Furthermore, let

$$A(r_k, \theta_l) \equiv \text{amplitude at range } r_k \text{ and bearing } \theta_l. \quad (\text{A3})$$

Let

$$N_{\max} = \frac{R_{\max}}{\Delta R} \quad (\text{A4})$$

where

$$N_{\max} = \text{maximum number of range cells used by radar} \quad (\text{A5})$$

and

$$R_{\max} = \text{maximum range unambiguously detected by radar.} \quad (\text{A6})$$

Let the sliding window be of length $2m + 3$. Numbering of range cells starts at the position plan indicator's center. The first range cell that can be tested is $k = m + 2$. The last range cell that can be tested is $k = N_{\max} - (m + 1)$. So the window can test range cells

$$\Gamma = \{m + 2, m + 3, \dots, N_{\max} - (m + 1)\}. \quad (\text{A7})$$

Consider the k^{th} test cell taken from the set Γ given in Eq. (A7) and let

$$j_{\text{lower-max}} \equiv \arg \max_j \{A(r_j, \theta_l) | j = k - 2, k - 3, \dots, k - (m + 1)\}, \quad (\text{A8})$$

$$j_{\text{upper-max}} \equiv \arg \max_j \{A(r_j, \theta_l) | j = k + 2, k + 3, \dots, k + (m + 1)\}, \quad (\text{A9})$$

$$A_{\text{lower-average}}(r_k, \theta_l) \equiv \frac{1}{m - 1} \sum_{\substack{1 \leq j \leq m \\ j \neq j_{\text{lower-max}}}} A(r_{k-1-j}, \theta_l), \quad (\text{A10})$$

$$A_{\text{upper-average}}(r_k, \theta_l) \equiv \frac{1}{m - 1} \sum_{\substack{1 \leq j \leq m \\ j \neq j_{\text{upper-max}}}} A(r_{k+1+j}, \theta_l), \quad (\text{A11})$$

and

$$A_{\text{max-average}}(r_k, \theta_l) \equiv \max[A_{\text{lower-average}}(r_k, \theta_l), A_{\text{upper-average}}(r_k, \theta_l)]. \quad (\text{A12})$$

The value passed by the DL is

$$A_{\text{passed-value}}(r_{k+p}, \theta_l) \equiv \chi[A(r_k, \theta_l) - A_{\text{max-average}}(r_k, \theta_l) - \tau_{\text{CFAR}}]A(r_{k+p}, \theta_l) + \chi[\tau_{\text{CFAR}} - \varepsilon - A(r_k, \theta_l) + A_{\text{max-average}}(r_k, \theta_l)]A_{\text{Thermal}} \quad (\text{A13})$$

where $k \in \Gamma$, the testable cells as given by Eq. (A7), and $\varepsilon > 0$ is related to the smallest non-zero A to D output. A convenient value of ε might be $\varepsilon = \ln(2)$ since the A to D converter maps signals into the integers 1 to 2047. Finally,

$$A_{\text{Thermal}} \equiv \text{amplitude due to thermal noise}, \quad (\text{A14})$$

$$\chi(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}, \quad (\text{A15})$$

$$\tau_{\text{CFAR}} \equiv \text{CFAR threshold}, \quad (\text{A16})$$

and

$$p \equiv -l, 0, l. \quad (\text{A17})$$

The index p in Eq. (A13) and Eq. (A17) takes into account not only the test cell k , but also the guard cells $k - l$ and $k + l$ to the right and the left of k .

Appendix B

OVERALL FITNESS, RULE FITNESS, INPUT-OUTPUT FITNESS, DATABASE STRUCTURE, AND UNCERTAINTY ANALYSIS FOR GP DATA MINING

All GPs must have a fitness function to evaluate the evolving population of chromosomes. The fitness function, referred to as the overall fitness, f_{OF} , is actually the sum of two other fitness functions. These functions are the rule fitness, f_{RF} , and the input-output fitness, f_{IOF} . The rule fitness is given in Eq. (B1), where the indicator function I_i is unity if the i^{th} rule in Table B1 is satisfied and is zero otherwise, and v_i is the value of the i^{th} rule as given in Table B1:

$$f_{RF} = \sum_{i=1}^{12} I_i \cdot v_i. \quad (\text{B1})$$

Table B1 is a rule summary having its origin in human expertise, common sense, economy, and other sources of information.

Let DL_j denote the j^{th} element of the evolving population of chromosomes within the GP for $j = 1, 2, \dots, m_{ps}$ where m_{ps} is the population size, i.e., the number of chromosomes. Let each DL_j consist of an *OR2* or *AND2* that connects two subgraphs, denoted as DL_j_left and DL_j_right . Let $l(DL_j_c)$ be the length, i.e., the number of nodes in DL_j_c , for $c \in \{left, right\}$. According to Table B1, if $l(DL_j_c)$ is greater than or equal to 20, then the parsimony pressure, $\alpha_p \cdot l(DL_j_c)$, is subtracted from the rule fitness followed by division by 100, ultimately yielding the rule score, denoted as g_{RS} . This subtraction is done if either $l(DL_j_left)$ or $l(DL_j_right)$ exceeds 20. The quantity α_p is referred to as the parsimony coefficient. The rule score can be expressed compactly as

$$\begin{aligned} g_{RS}(DL_j) = & \frac{c}{100} \left\{ f_{RF}(DL_j_left) - \chi[l(DL_j_left) - 20] \cdot \alpha_p \cdot l(DL_j_left) \right\} \\ & + \frac{c}{100} \left\{ f_{RF}(DL_j_right) - \chi[l(DL_j_right) - 20] \cdot \alpha_p \cdot l(DL_j_right) \right\}. \end{aligned} \quad (\text{B2})$$

The parameter c in Eq. (B2) is the product of the indicator functions for rules 14–19 in Table B1 for the closed-form results in this section, and c is the product of the indicator functions for rules 14 and 15 for the computational results in section B2.

If the rule score exceeds the rule threshold denoted as κ_{RT} , then and only then is the input-output fitness evaluated. By forcing the rule score to exceed a threshold before the input-output fitness is evaluated, a great deal of computational complexity is avoided.

Table B1 — Rule Set for Closed-Form Results and Computational Genetic Programming Experiments

RULE	RULE SUMMARY
R1	If either <i>OR3DELAY</i> or <i>AND3DELAY</i> are present during rule fitness evaluation, add $v_1 = 5$.
R2	If R1 is satisfied and <i>OR3DELAY</i> or <i>AND3DELAY</i> are at the end during rule fitness evaluation, add $v_2 = 6$.
R3	If <i>AND3</i> or <i>OR3</i> are present during fitness evaluation, add $v_3 = 5$.
R4	If none of the three comparators denoted as $H_i, i = 1, 2, 3$ follow <i>AND3</i> or <i>OR3</i> during fitness evaluation, add $v_4 = 3$.
R5	If there is only one delay present, i.e., in the form of <i>OR3DELAY</i> or <i>AND3DELAY</i> , during rule fitness evaluation, add $v_5 = 5$.
R6	<p>Let</p> <p><i>have_not_alone_signal</i> = the number of times that <i>SUM_SIG_i</i> or <i>MAX_SIG_i</i>, $i = \{1, 2, 3\}$ or 2 or 3 is fed into $H_i, i = 1, 2, 3$ or <i>DIFF</i>.</p> <p>Let</p> <p><i>num_signals</i> = the number of times <i>SUM_SIG_i</i> or <i>MAX_SIG_i</i> for $i = \{1, 2, 3\}$ or 2 or 3 appear in the chromosome.</p> <p>During rule fitness evaluation add $v_6 = 5 \cdot \left(\frac{\text{have_not_alone_signal}}{\text{num_signals}} \right)$.</p>
R7	If none of the $H_i, i = 1, 2, 3$ are adjacent to each other, then during fitness evaluation add $v_7 = 5$.
R8	<p><i>no_diff2H</i> = number of times a <i>DIFF</i> feeds into a Heaviside step function.</p> <p><i>no_diff</i> = number of <i>DIFF</i> operators in the DL. During rule fitness evaluation add</p> <p>$v_8 = 5 \cdot \left(\frac{\text{no_diff2H}}{\text{no_diff}} \right)$.</p>
R9	<i>no_diffw2sig</i> = number of times two consecutive signals feed into a <i>DIFF</i> . During rule fitness evaluation add $v_9 = 5 \cdot \left(\frac{\text{no_diffw2sig}}{\text{no_diff}} \right)$.
R10	<i>no_diffw2diffsig</i> = number of times a <i>DIFF</i> takes two different signals. During rule fitness evaluation add $v_{10} = 5 \cdot \left(\frac{\text{no_diffw2diffsig}}{\text{no_diff}} \right)$.
R11	<p><i>no_maxorsum</i> = number of times <i>DIFF</i> takes two <i>MAX_SIG</i> input or two <i>SUM_SIG</i> inputs.</p> <p>During rule fitness evaluation add $v_{11} = 5 \cdot \left(\frac{\text{no_maxorsum}}{\text{no_diff}} \right)$.</p>
R12	During rule fitness evaluation add $v_{12} = 5 / \text{no_diff}$.
R13	Penalize the rule fitness by subtracting parsimony pressure if the chromosome length is equal to or greater than 20 and divide the difference by 100.

Table B1 (continued) — Rule Set for Closed-Form Results and Computational Genetic Programming Experiments

RULE	RULE SUMMARY
R14	<p>There are two subgraphs of the same form. One subgraph of the DL uses <i>SIG2</i> and <i>SIG123</i> as input; the other side uses <i>SIG3</i> and <i>SIG123</i> as input, i.e.,</p> $subgraph1 = DC_3 H_m diff A_{\{2\}}^{\sigma} A_{\{1,2,3\}}^{\sigma} H_q A_{\{2\}}^{\alpha} H_r A_{\{1,2,3\}}^{\beta}$ <p>and</p> $subgraph2 = DC_3 H_m diff A_{\{3\}}^{\sigma} A_{\{1,2,3\}}^{\sigma} H_q A_{\{3\}}^{\alpha} H_r A_{\{1,2,3\}}^{\beta}$ <p>If this rule is satisfied then $I_{14} = 1$, otherwise $I_{14} = 0$.</p>
R15	<p>Join the subgraphs using <i>OR2</i> or <i>AND2</i>, i.e.,</p> $z = C_2 DC_3 H_m diff A_{\{2\}}^{\sigma} A_{\{1,2,3\}}^{\sigma} H_q A_{\{2\}}^{\alpha} H_r A_{\{1,2,3\}}^{\beta} DC_3 H_m diff A_{\{3\}}^{\sigma} A_{\{1,2,3\}}^{\sigma} H_q A_{\{3\}}^{\alpha} H_r A_{\{1,2,3\}}^{\beta}$ <p>If this rule is satisfied then $I_{15} = 1$, otherwise $I_{15} = 0$.</p>
R16	<p>The threshold τ_2 is used after the <i>DIFF</i>.</p> $z = C_2 DC_3 H_2 diff A_{\{2\}}^{\sigma} A_{\{1,2,3\}}^{\sigma} H_q A_{\{2\}}^{\alpha} H_r A_{\{1,2,3\}}^{\beta} DC_3 H_2 diff A_{\{3\}}^{\sigma} A_{\{1,2,3\}}^{\sigma} H_q A_{\{3\}}^{\alpha} H_r A_{\{1,2,3\}}^{\beta}$ <p>If this rule is satisfied then $I_{16} = 1$, otherwise $I_{16} = 0$.</p>
R17	All thresholds must be used. If this rule is satisfied then $I_{17} = 1$, otherwise $I_{17} = 0$.
R18	Assume that $\beta \neq \sigma$ and $\alpha = \sigma$. If this rule is satisfied then $I_{18} = 1$, otherwise $I_{18} = 0$.
R19	Each subgraph should be as short as is consistent with the above rules. If this rule is satisfied then $I_{19} = 1$, otherwise $I_{19} = 0$.

Let DL^T denote the true DL diagram that underlies the sensor device used to construct the input-output database. For the examples considered in this paper, let there be three signals. The input-output database is assumed to have the following structure:

$$M_{DB} = \begin{bmatrix} \vec{S}_1^1 & \vec{S}_2^1 & \vec{S}_3^1 & B^1 \\ \vec{S}_1^2 & \vec{S}_2^2 & \vec{S}_3^2 & B^2 \\ \vdots & \vdots & \vdots & \vdots \\ \vec{S}_1^m & \vec{S}_2^m & \vec{S}_3^m & B^m \end{bmatrix}. \quad (B3)$$

Where \vec{S}_j^k is the three-time-window input from the j^{th} source for the k^{th} input, $B^k \in \{0,1\}$ is the k^{th} output from DL^T for $k=1,2,\dots,m$; i.e.,

$$B^k = DL^T \left(\vec{S}_1^k, \vec{S}_2^k, \vec{S}_3^k \right); \quad \text{for } k = 1, 2, \dots, m. \quad (B4)$$

The input-output fitness for the j^{th} chromosome, DL_j , is defined as

$$f_{IOF}(j, M_{DB}) = 1 / \left[1 + \sum_{k=1}^m \left| DL_j(\vec{S}_1^k, \vec{S}_2^k, \vec{S}_3^k) - B^k \right|^2 \right]. \quad (B5)$$

The overall fitness, f_{OF} , for the j^{th} chromosome can be written as

$$f_{OF}(j, M_{DB}) = g_{RS}(DL_j) + \chi(g_{RS}(DL_j) - \kappa_{RT}) \cdot f_{IOF}(j, M_{DB}). \quad (B6)$$

It is important to recall that in actual implementation, the input-output fitness is only evaluated if the rule fitness is greater than or equal to the rule threshold. Selectively evaluating the input-output fitness greatly reduces the computational complexity and hence the run time of the GP.

B1. Closed-Form Results for the Class of DL Maps and Database that Maximize the Various Fitness Functions

It is possible through inspection of Eq. (B2) to write down a set, denoted as Z_{MLDL} , of minimum length DL maps that result in a global maximum for the rule score. Analysis of Z_{MLDL} makes it possible to also write down a closed-form expression for the image set, denoted as R_{MLDL} , under the DLs making up Z_{MLDL} . The elements of R_{MLDL} can be written as explicit functions of the parameters that characterize distinct elements of Z_{MLDL} . From the explicit form of R_{MLDL} it is possible to invert out a database matrix with closed-form expressions for its elements. The closed-form expressions are given in terms of sensor system parameters. Table B2 gives a small subset of a database created in this fashion. A database given explicitly in terms of the sensor system parameters can be used in uncertainty analysis and GP convergence tests. It is readily shown using the rule set and the closed-form database that emerges from this approach that loss of rules or rows of the input-output database can result in not one but many DLs that maximize the overall fitness.

Table B2 — Parameters for a Subset of the Input-Output Database Derived from the Rules in Table B1

Input-Output Table for Three-Signal Example									
Component	γ_1^q	γ_2^q	γ_3^q	$\gamma_1^{q'}$	$\gamma_2^{q'}$	$\gamma_3^{q'}$	Window2	Window3	Output
OR2	10^{τ_3}	1	$-2 \cdot 10^{\tau_3}$	0	0	0	$\bar{O}(N)$	$\bar{O}(N)$	1
OR3DELAY	10^{τ_3}	-10^{τ_3}	-10^{τ_3}	0	0	0	$\bar{O}(N)$	$\bar{O}(N)$	1
AND3	1	1	10^{τ_3}	0	0	0	$\bar{O}(N)$	$\bar{O}(N)$	0
AND3	10^{τ_3}	1	1	0	0	0	$\bar{O}(N)$	$\bar{O}(N)$	0
τ_3 along Auxillary Antenna	10^{τ_1}	10^{τ_1}	-10^{τ_3}	0	0	0	$\bar{O}(N)$	$\bar{O}(N)$	1

Preliminary to the development described above it is useful to define the following subsets of the terminal and function sets. Let

$$D \in DELAY = \{AND3DELAY, OR3DELAY\}, \quad (B7)$$

$$C_3 \in CONN3 = \{AND3, OR3\}, \quad (B8)$$

$$C_2 \in CONN2 = \{AND2, OR2\}, \quad (B9)$$

$$COMP = \{H_1, H_2, H_3\}, \quad (B10)$$

and

$$A_a^\sigma, A_b^\sigma, A_i^\alpha, A_j^\beta \in MAXSUM = \{MAX_SIG1, MAX_SIG2, MAX_SIG3, MAX_SIG123, \dots, SUM_SIG1, SUM_SIG2, SUM_SIG3, SUM_SIG123\}. \quad (B11)$$

The parameters σ , α , and β carry the value *MAX* or *SUM* and a, b, i and j take the values $\{1\}, \{2\}, \{3\}$, and $\{1, 2, 3\}$. As examples of this notation, $A_{\{3\}}^{MAX}$ corresponds to *MAX_SIG3*, whereas $A_{\{1,2,3\}}^{SUM}$ corresponds to *SUM_SIG123*.

The subclass of DLs arising from rules 1–12 that are of minimum length and maximize the rule score for that length take the following form:

$$z(D, C_3, a, b, i, j, \sigma, \alpha, \beta, l, q, r) = DC_3 H_l DIFF A_a^\sigma A_b^\sigma H_q A_i^\alpha H_r A_j^\beta, \quad (B12)$$

where

$$a \neq b \quad a, b = \{1\}, \{2\}, \{3\}, \{1, 2, 3\}; \quad l, q, r = 1, 2, 3. \quad (B13)$$

Rules 1–13 govern only half the DL design, i.e., one subgraph of the tree. It follows from rule 14 that the other subgraph of the DL map will also take a form given by Eq. (B12). It follows from rules 15–19 that the DL map that includes both subgraphs takes the form

$$Z(D, C_3, \sigma, \sigma, \beta, 2, q, r) = C_2 z(D, C_3, \{2\}, \{1, 2, 3\}, \{2\}, \{1, 2, 3\}, \sigma, \sigma, \beta, 2, q, r) \cdot z(D, C_3, \{3\}, \{1, 2, 3\}, \{3\}, \{1, 2, 3\}, \sigma, \sigma, \beta, 2, q, r), \quad (B14)$$

where

$$\sigma \neq \beta. \quad (B15)$$

Also, from rule 17, all thresholds must be used, so since threshold two has been used already, once one additional threshold is assigned, the other is known.

Equation (B14) gives a typical element of Z_{MLDL} . The use of all 19 rules in Table B1 reduces Z_{MLDL} to a set with 32 elements. It would hardly be worth using a GP for such a small number of alternative solutions. The 19 rules of Table B1 allow simple closed-form results for the elements of Z_{MLDL} . Likewise, the simplicity of Z_{MLDL} arising from the 19 rules allows a database to be inverted in closed

form. The overall fitness will have a single global maximum at the desired DL map, i.e., DL^T . The database subset given in Table B2, taken from a much larger database, is also a closed-form result, exhibiting the database matrix's explicit dependence on system parameters, i.e., thresholds. Table B2 is a subset of a database constructed using only rules 1–15 from Table B1.

The first and one of the most important steps in data mining is the construction of the database. The database is generally constructed by, or at least “cleansed” by, a domain expert so that it reflects truth. For this example it is valuable to construct an ideal database. This database can actually be derived mathematically and rendered in closed form as explicit functions of the system parameters and DL graph labels that must be determined. A database of this kind is very effective in studying the effect of uncertainty on the ultimate DL evolved. For the analysis in this report, uncertainty refers to the loss of a rule or rules from Table B1 or the loss of at least one row from the database matrix given in Table B2.

To understand the database subset, it is essential to define some notation. Let

$$\bar{\Gamma}_i^q \equiv \left[\gamma_i^q, \gamma_i^{q'}, \bar{O}(N-2) \right]; \quad \text{for } i=1,2,3; \quad q=1,2,\dots,m; \quad (\text{B16})$$

where γ_i^q and $\gamma_i^{q'}$ are the first and second pulse values for the signal denoted by $\bar{\Gamma}_i^q$. The quantity $\bar{O}(n)$ is defined to be a row vector consisting of n zeros. The quantity $\bar{\Gamma}_i^q$ is the i^{th} source signal for the q^{th} measurement to be stored in the database, i.e., the q^{th} row in the database matrix Eq. (B3) for one time window.

Table B2 provides a subset of a database derived by solving an analog of Eq. (B14) for rules 1-15, by inspection or solving systems of inequalities arising from the analog of Eq. (B14). Each row of Table B2 is determined to encourage the GP to place a particular label on the graph, the label of interest being given in the first column of that row. As indicated in the first row, the second through seventh columns give the values of γ_i^q and $\gamma_i^{q'}$ for signals $i=1,2$, and 3 . The entry $\bar{O}(N)$ implies there are N zeros in the window. There is no need to have separate entries for each source signal for windows two and three as all three signals exhibit the same behavior, i.e., they have zero entries for the final two windows. The final column gives the simulated measured output of the system of interest.

Also, under ideal conditions there is great freedom in specifying input. When making measurements using a real sensor device, the freedom to specify any desired input does not exist. The sensor device will only accept certain input, and the output is likely to be corrupted by noise and might be characterized by lost data. Given these difficulties, a real database must be much larger to be able to deal with corruption and because the observer cannot select the most desirable input-output entries.

The rule set in Table B1 and database in Table B2 have value for understanding the nature of uncertainty in expertise and data. If certain rules are removed from Table B1, the number of potential DL maps grows rapidly. For example, if Rule 17 is eliminated from Table B1, then between the remaining rules and the database there is not enough information to uniquely specify the threshold labeling. So DL^T is no longer uniquely determined. The resulting DL maps will have underlying graphs that in graph-theoretic terms are isomorphic to the graph of DL^T . An isomorphism exists between two graphs if there exists a map that establishes a one-to-one correspondence between the vertices of the two graphs and the map preserves edges. If the goal of the observer is to determine a signal that will produce a certain response from the sensor device, then it is often sufficient to obtain DL maps that differ only in threshold labeling. If the observer knows the maximum possible threshold for the system, but does not know which thresholds label particular nodes, he can in many instances create a signal that gives the desired response

by simply increasing the energy in the signal to correspond to the potential of the maximum threshold labeling a particular node. So in this case, the rule and the input-output database uncertainty are not extreme liabilities. The input signal's functional form does not change; the true cost of uncertainty is a minor increase in the energy contained in the signal.

B2. DATA MINING RESULTS

In this section, two different DL schemes data mined by the GP are considered. These two examples are representative of the many experiments that have been conducted to show the effectiveness of the GP-based data mining procedure presented in this paper. The first example is the DL represented in Eq. (14) and in Fig. 1. This DL will assume the value of DL^T for the discussion below. Using various databases too large to reproduce here and different random number generator seeds, the GP was able to reverse engineer Eq. (14) in no more than 76 GP generations. The different number of generations and amounts of CPU time required reflect the effect of different input-output databases and also the random number generator seeds. One database may constrain the evolutionary process more than another, resulting in fitness values that over time push the population more rapidly toward DL^T . Also, since the initial population is generated randomly, and crossover, mutation, architecture-altering steps (AAS), and symmetrical replication (SR) have random aspects, a change in the seed of the random number generator can also impact run time.

To get a feel for the evolutionary process it is useful to examine some intermediate generations that lead to DL^T . For the case in which Eq. (14) is reverse engineered in 76 generations, the elite chromosomes found for different generations are provided in Table B3, with the 76th generation reproducing the correct chromosome as given in Eq. (14).

Table B3 — Evolution of the DL Depicted in Figure 1

Generation	Best Chromosome found in the Population for the Indicated Generation
1	$OR2\ H_2\ DIFF\ SUM_SIG123\ MAX_SIG123\ OR2\ DIFF\ MAX_SIG2\ H_1\ MAX_SIG2\ SUM\ SIG3$
25	$OR2\ AND3DELAY\ OR3\ H_3\ MAX_SIG2\ H_2\ SUM_SIG123\ H_2\ DIFF\ MAX_SIG2\ MAX_SIG123\ AND3DELAY\ OR3\ H_2\ SUM_SIG2\ SUM_SIG3\ H_2\ DIFF\ MAX_SIG2\ MAX\ SIG123$
40	$OR2\ OR3DELAY\ AND3\ H_1\ MAX_SIG2\ H_1\ MAX_SIG123\ H_2\ DIFF\ SUM_SIG2\ SUM_SIG123\ AND3DELAY\ AND3\ H_3\ SUM_SIG3\ H_1\ SUM_SIG2\ H_2\ DIFF\ SUM_SIG2\ SUM\ SIG3$
50	$OR2\ OR3DELAY\ AND3\ H_1\ MAX_SIG2\ H_1\ MAX_SIG123\ H_2\ DIFF\ SUM_SIG2\ SUM_SIG123\ OR3DELAY\ AND3\ H_1\ MAX_SIG3\ H_1\ MAX_SIG123\ H_2\ DIFF\ SUM_SIG3\ SUM\ SIG123$
76	$OR2\ OR3DELAY\ AND3\ H_1\ MAX_SIG123\ H_3\ SUM_SIG2\ H_2\ DIFF\ SUM_SIG2\ SUM_SIG123\ OR3DELAY\ AND3\ H_1\ MAX_SIG123\ H_3\ SUM_SIG3\ H_2\ DIFF\ SUM_SIG3\ SUM\ SIG123$

The chromosomes entered into Table B3 reflect some of the characteristics observed during the evolutionary process. From the first generation forward, the GP is able to find best candidates that have an $OR2$ at the end of the chromosome. This property arises from rule 15 in Table B1. The presence of two

DIFF operators in the first generation is also promising. The best chromosome for the first generation is much too short when compared to the desired result.

New innovations are found in generation 25 in that both arguments of both *DIFF*s use *MAX* functions as well as the *SIG123* structure. Even though it is expected that both arguments will ultimately use *SUM* functions, the use of a common function for both arguments may show evolution in the proper direction. Both *DIFF* operators are preceded by H_2 which is what is found in Eq. (14). Even with these innovations, the best chromosome of generation 25 is far from the correct result.

The best chromosome of the 40th generation has subgraphs consistent with the closed-form result of Eq. (B12). All generations after the 26th have elite chromosomes that assume this form. The computational results use a smaller set of rules, i.e., rules 1–15, than those used in deriving Eq. (B13) through Eq. (B15), so the rules alone do not require that the solutions converge to this form. The GP's effort from generation 27 through 76 involves finding a solution with the proper node labels. Various rows in the input-output database contribute to proper labeling; for example, if the fifth row in the database subset in Table B2 is deleted, it is likely that the final GP solution will not have proper threshold labeling. An improper threshold value is undesirable from the standpoint of trying to reproduce the exact DL. However, if the goal is to produce an input signal that produces unity as an output, then even with the wrong threshold value, the desired output is obtained as long as the input signal has sufficient energy to take into account uncertainty. In conclusion, the ultimate cost of information uncertainty in this case is a small amount of additional energy.

The best chromosome of the 50th generation is far closer to Eq. (14). The *MAX* functions in the arguments of the *DIFF* have been replaced by *SUM* functions. The arguments of the *DIFF* operators are the ones for the final result and the output of both *DIFF*s is passed into H_2 as found in Eq. (14). In this chromosome, replacing $H_1 \text{ MAX_SIG2}$ with $H_3 \text{ SUM_SIG2}$ and $H_1 \text{ MAX_SIG3}$ with $H_3 \text{ SUM_SIG3}$ would yield the correct result. Finally, the desired result is found in generation 76.

For the second example, consider the DL given below in Eq. (B17) as DL^T , i.e., truth:

$$\begin{aligned} &OR2 \ OR3DELAY \ AND3 \ H_1 \ SUM_SIG123 \ H_3 \ MAX_SIG2 \ H_2 \ DIFF \ MAX_SIG2 \ MAX_SIG123 \\ &OR3DELAY \ AND3 \ H_1 \ SUM_SIG123 \ H_3 \ MAX_SIG3 \ H_2 \ DIFF \ MAX_SIG3 \ MAX_SIG123. \end{aligned} \quad (B17)$$

The GP's evolutionary process for inverting Eq. (B17) is summarized in Table B4.

This example is similar to Eq. (14); in fact if in Eq. (14) the *MAX* operations are replaced by the *SUM* operation and *SUM* replaced by *MAX* then Eq. (B17) is obtained. Given that Eq. (14) and Eq. (B17) differ only in labeling of the underlying graph, it is anticipated that the GP-based evolutionary processes that yield Eq. (14) and Eq. (B17) will be similar. This anticipation is borne out, but there are some differences in the evolutionary processes. One significant difference is that the chromosome in Eq. (B17) is evolved in a smaller number of generations than the one in Eq. (14). There is nothing obvious about the rule set or input-output database used for both chromosomes that would favor one over the other. Experimentation seems to indicate that the difference in the number of generations required is related to the seed of the random number generator.

Table B4 — Evolution of the DL given in Eq. (B17)

Generation	Best Chromosome found in the Population for the Indicated Generation
1	<i>OR2 SUM SIG2 AND3DELAY DIFF SUM SIG2 SUM SIG123</i>
8	<i>OR2 AND3DELAY DIFF SUM SIG3 SUM SIG123 AND3DELAY H₂ DIFF MAX SIG2 MAX SIG123</i>
16	<i>OR2 AND3DELAY OR3 H₂ SUM SIG2 SUM SIG3 H₂ DIFF MAX SIG2 MAX SIG123 OR3DELAY H₂ OR3 H₂ SUM SIG2 SUM SIG3 H₂ DIFF MAX SIG2 MAX SIG123</i>
30	<i>OR2 OR3DELAY AND3 H₂ MAX SIG123 H₃ MAX SIG2 H₂ DIFF MAX SIG2 MAX SIG123 AND3DELAY AND3 H₃ SUM SIG2 H₁ SUM SIG123 H₂ DIFF SUM SIG2 SUM SIG3</i>
46	<i>OR2 OR3DELAY AND3 H₁ SUM SIG123 H₃ MAX SIG2 H₂ DIFF MAX SIG2 MAX SIG123 OR3DELAY AND3 H₁ SUM SIG123 H₃ MAX SIG3 H₂ DIFF MAX SIG3 MAX SIG123</i>

Just as with the example in Table B3, the best chromosome of the first generation has an *OR2* at the end, but is otherwise too short and far removed from the correct answer. By the eighth generation the *H₂DIFF MAX_SIG2 MAX_SIG123* structure has emerged. The best chromosome of the 16th generation preserves the best features of previous generations and also makes use of an *OR3DELAY*, but still has many defects. For all generations after the 26th generation, the elite chromosome has two subgraphs that take the form found in closed form in Eq. (B12). The elite chromosome of the 30th generation has many correct labels and incorrect ones. It illustrates how evolution can fluctuate from generation to generation, producing individuals of higher fitness, but departing significantly from the true DL in form. Finally, in generation 46 the GP converges, having produced the correct DL design.